One Size Does Not Fit All:
Towards User & Query Dependent Ranking For
Web Databases

Aditya Telang and Chengkai Li and Sharma Chakravarthy

Technical Report CSE-2009
One Size Does Not Fit All: Towards User & Query Dependent Ranking For Web Databases

Aditya Telang, Chengkai Li, Sharma Chakravarthy

Department of Computer Science and Engineering, University of Texas at Arlington
aditya.telang@mavs.uta.edu, cli@uta.edu, sharmac@uta.edu

July 16, 2009

Abstract

In this paper, we propose an automated solution for ranking query results of Web databases in an user- and query-dependent environment. We first propose a learning method for inferring a workload of ranking functions by investigating users’ browsing choices over individual query results. Based on this workload, we propose a similarity model, based on two novel metrics – user- and query-similarity, for ranking query results when user browsing choices are not available. We present the results of an experimental study that validates our proposal for user- and query-dependent ranking.

1 Introduction

With the emergence of the deep Web, a large number of Web databases and their related applications (e.g., airline reservations, vehicle search, real estate scouting, ...) have proliferated. Since these databases support a simple Boolean query retrieval model, it often leads to situations when too many results are generated in response to a query. In order to find the results that match one’s interest, the user has to browse through this large result set – a tedious and time-consuming task. Currently, Web databases simplify this task by displaying these results in a sorted order on the values of a single attribute (e.g., Price, Mileage, etc.). However, most Web users prefer to simultaneously examine multiple attributes and their values (instead of a single attribute) while selecting results relevant to them, and these preferences vary as the user queries change.

Consider Google Base’s [1] Vehicle database that comprises of a table with attributes such as Make, Price, Mileage, Location, Color, etc. where each tuple represents a vehicle for sale in the US, and note the following two scenarios that we use as running examples for the rest of the paper –

Example-1: Two users – a company executive (U_1), and a student (U_2) seek answers to a query (Q_1) – “Make=Honda And Location=Dallas, TX”. In response to this query, over nineteen thousand tuples are generated by Google Base. Intuitively, U_1 would search for new vehicles with specific color choices (e.g., only red colored vehicles).
Hence, the ranking function for $U_1$ should rank vehicles with “Condition=New And Color=Red” higher than the others. In contrast, $U_2$ would most likely search for old vehicles without having any specific preferences on other attributes of the vehicles. Consequently, $U_2$’s ranking function should display vehicles with “Condition=Old” ranked higher than the others.

**Example-2:** The same student user ($U_2$) moves to a different university (in “Seattle, WA”) on a scholarship and asks a different query ($Q_2$) – “Make=Toyota And Location=Seattle,WA”. Although, he may still be looking for used vehicles, we can presume (since he has attained a scholarship) that he may be willing to pay a higher price for a lesser mileage vehicle (e.g., “Mileage<100,000”) instead of settling for a vehicle with any mileage value. His preferences for $Q_2$ would then require ranking vehicles with “Condition=Old And Mileage<100,000” higher than others.

**Example-1** illustrates that different Web users may have contrasting ranking preferences over the results of the same query. Similarly, **Example-2** emphasizes that the same user may possess distinct ranking preferences for different queries (at different points in time). Thus, it is evident that in a setting (such as Web databases) where a large set of queries given by a varied set of users is involved, the corresponding results should be ranked in a user- and query-dependent manner.

The current sorting mechanism adopted by Web databases does not hold the ability to perform such ranking. Some of the proposed extensions to SQL [2][3] allow manual specification of attribute weights (and thus, the ranking function), an approach cumbersome for most Web users. The use of domain expertise for estimating these attribute weights has also been suggested [4]; however, the problem being that domain experts establish a relative ordering between the attributes (e.g., “Price” and “Mileage” are more important than “Color” in the domain of vehicles) without estimating the absolute attribute weights and the preferences for attribute values (e.g., one user may prefer ‘red’ colored car whereas another may prefer ‘blue’ colored car) that can translate to corresponding ranking functions. Automated ranking of database results has been studied in the context of relational databases [5][6][7][8] and the most commonly proposed technique is to derive a ranking function that is either user-independent or query-independent (or both i.e., a single function) for ordering the tuples. In the context of Web databases, as motivated by the above examples, an ideal ranking model should consider both the dimensions – user and query in conjunction, to form a robust framework for ranking.

In this paper, we propose such a user- and query-dependent approach for ranking Web database query results. We infer the ranking function for an user query via a learning technique that analyzes the user’s browsing choices over the query results. Unlike relational databases, the nature of Web database applications allows users to browse and select the results that match their preferences (through an interaction with the Web pages containing the result tuples). Hence, although an user’s explicit ranking preferences over the results of a query are not available, we believe that the results chosen by an user aid in implicitly indicating his/her ranking preferences from which the corresponding ranking function can be derived. In the case of Example-1 for instance, $U_1$ may select (or click) those tuples which represent ‘new’ and ‘red’ colored vehicles. In contrast, $Q_1$’s actual results will comprise ‘new’ and ‘old’ vehicles with different colors (‘red’, ‘blue’, ‘green’, ...). By analyzing the relationship between these two sets
of results, $U_1$’s ranking preferences for $Q_1$ can be derived.

The above intuition is translated into a principled ranking function that is represented as a \textit{linear weighted-sum} function comprising of – i) \textit{attribute-weights}, that denote the significance associated with individual attributes, and ii) \textit{value-scores}, that divulge the importance of the the attribute values in the chosen results. We propose a novel learning technique, \textit{Probabilistic Data Distribution Difference}, to deduce these \textit{attribute-weights} and \textit{value preferences}.

In a general setting, it is impossible to procure the browsing choices for every user over the results of every possible query, and hence, inferring the corresponding ranking functions will be infeasible (e.g., $U_2$’s preferences towards $Q_2$’s results may be unknown apriori). In order to tackle this issue, we propose a novel \textit{similarity}-based technique based on the intuition that – i) similar users, for the same query, tend to display similar ranking preferences, and ii) the same user tends to display similar ranking preferences over the results of similar queries.

We formalize these notions of similarity into – 1) \textit{user similarity}, a metric of similarity between users, estimated based on their past browsing choices, and 2) \textit{query similarity}, a metric of similarity between queries, estimated using two independently proposed measures – i) \textit{query-condition similarity}, and ii) \textit{query-result similarity}. The former establishes query similarity by equating the conditions in the respective queries, whereas the latter establishes it by collating their corresponding results.

For a Web database, the scenario of two similar users asking the same query (an user $U_1$ and $U_2$ asking $Q_2$) and/or the same user asking two similar queries ($U_2$ asking a query $Q_j$ and $Q_2$) is remote. However, the likelihood that a similar user ($U_i$) asked a similar query ($Q_j$) in the past, for which a ranking function was inferred, is relatively high. Based on this setting, we combine the above two metrics into a single \textit{Similarity Model} to determine the \textit{most similar query} asked by the \textit{most similar user} from a workload of past user queries (for which functions are inferred based on browsing choices).

\textbf{Contributions:} The contributions of this paper are:

1. We propose an automated solution for ranking Web database query results in a user- and query-dependent environment.

2. Our proposed approach derives the ranking function for an user query by analyzing the user’s browsing choices over the query results using a learning technique.

3. We propose a \textit{Similarity Model}, based on two novel measures – \textit{query similarity} and \textit{user similarity}, to derive ranking functions for user queries whose browsing choices are unknown at query time.

4. We report on an elaborate set of experimental results that validate our claim of user- and query-dependent ranking for Web databases, and establish the effectiveness of deriving ranking functions from browsing choices and similarity-based techniques.

\textbf{Roadmap:} In Section 2, we discuss the related work. In Section 3, we formally define the ranking problem, clearly outline the sub-problems embedded within it, and describe the architecture of our ranking system. Section 4 discusses our approach of deriving a
ranking function for a user query based on the user’s browsing choices, and Section 5 explains our Similarity Model. In Section 6, we discuss the results of our experiments, and Section 7 concludes the paper.

2 Related Work

Over the past years, there has been significant interest in ranking query results for relational databases. For instance, the work in [5] proposes ranking for database tuples in a query- and user-independent framework. This model relies on the availability of a workload of queries spanning all attributes and values to establish a score for a tuple. However, a drawback of such a workload is that in the context of Web databases, user queries are restricted to a subset of the attributes that are displayed in the results. In such a setting, the workload will fail to capture user preferences towards those attributes and values that cannot be specified in the query. In contrast, we capture these preference via users’ browsing choices in a query- and user-dependent setting.

The work proposed for query-dependent ranking [7] analyzes the relationship between the query results and the tuples in the database. However, a major drawback of this work lies in the fact that it requires the knowledge of the complete underlying database at all times to rank query results, an improbable setting for Web databases that dynamically obtain data from a slew of individual sources. In contrast, we establish query-dependent ranking by analyzing the user’s browsing choices and comparing different queries in terms of their similarity with each other without requiring full knowledge of the Web database.

Context preferences for user-dependent ranking have been proposed in [8], [9] and [10]. However, these models require the user to specify an order/preference for the tuples in the absence of a specific query from which a global ordering across the database is obtained. In [2], [3], and [11], the SQL query language is extended to allow the user to specify the ranking function according to their preference for the attributes. In [11], [12], and [13], user relevance feedback is employed to learn the similarity between a result record and the query for ranking in relational multimedia databases. However, all these approaches require considerable user input and is an arduous task for Web users who have no clear idea how to assign order to tuples and/or attributes. In contrast, our proposal relies purely on users’ browsing choices that reveal their implicit ranking preferences without requiring them to have an excessive interaction with the system.

The use of query similarity has been widely studied in Information Retrieval and Collaborative Filtering. However, as shown in [14], database queries involving multiple combinations cannot be directly compared like IR-keyword queries. In this paper, we propose a novel notion of database query similarity that is determined by analyzing the results for individual queries. Although, an intuitive mechanism for establishing user similarity based on profiles has been studied in [4], and [15], it involves the use of domain experts in addition to learning models to derive this similarity. Alternatively, we propose a mechanism to capture user similarity by analyzing the relationship between the users’ past browsing choices.

The use of learning methods for deriving ranked lists has been studied extensively
in Machine Learning and Image Processing [16], [17], [18]. Similar to these techniques, we propose a probabilistic learning method for capturing attribute preferences for Web queries. Our results show that within the framework that we tested, our proposed model performed better than existing bayesian or regression models. However, there exists a number of sophisticated learning models that can aid in inferring a ranking function, but comparing them has not been the major focus of this work. To the best of our knowledge, user- and query-dependent ranking in the context of Web database queries has not been studied in literature.

3 Problem Definition and Architecture

We now formally define the problem of ranking in Web databases, and outline a general architecture of our solution.

3.1 Problem Definition

Consider a Web database table $D$ over a set of $m$ attributes, $A = \{A_1, A_2, ..., A_m\}$. An user $U$ asks a query $Q$ of the form – “SELECT * FROM $D$ WHERE $X_1 = x_1$ AND $\cdots$ AND $X_s = x_s$”, where each $X_i \in A$ and $x_i$ is a value in its domain. Let $N = \{t_1, t_2, ..., t_n\}$ be the set of result tuples for $Q$.

The ranking problem can be stated as: “For a query $Q$ given by user $U$, determine a ranking function $F_{UQ}$ that assigns a score to every tuple from which a ranking order for $N$ can be established”. As spelled out in Section 1, we categorize this problem into two individual sub-problems:

1) **Inferring Ranking Functions using Browsing Choices:** Given an user $U$, the query $Q$, and the set of results $N$, let $R (\subset N)$ be the set of results generated based on $U$’s browsing choices over $N$. The ranking problem can then be stated as: “Using $R$ and $N$, determine the ranking function $F_{UQ}$ that captures $U$’s preferences over $Q$’s results.”

2) **Inferring Ranking Functions using Similarity Measures:** Consider that an user’s ($U$) browsing choices for the results of a query ($Q$) are not available. In this setting, the ranking problem is stated as: “Assuming a workload of past user queries for which the ranking functions are derived apriori, determine the ranking function ($F_{U_iQ_j}$) of the most similar user ($U_i$) to $U$, derived for the most similar query ($Q_j$) to $Q$, for ranking $Q$’s results.”

The above scenario only represents the simplest problem instance. For example, the type of queries described above are point queries since they specify single-valued equality conditions on each of the specified attributes. In a more general setting, queries may contain range/IN conditions, and/or Boolean operators other than conjunctions. However, in this paper, we focus only on the problem of point queries over a single table of a Web database.
3.2 Ranking Architecture

The architecture for our user- and query-dependent ranking framework (shown in Figure 1) comprises of two components for addressing the subproblems defined above.

In the first component (Inferring via Browsing Choices), the user’s ($U$) browsing choices over the query results ($N$) produces the set of relevant results ($R$). Both these sets ($N$ and $R$) are fed to the Learning Model that deduces the – i) significance of each attribute to establish the set of attribute-weights, and ii) emphasis given by users to particular values of an attribute. Based on the preferences associated with the values, a normalization scheme translates the values into their corresponding value-scores. The attribute-weights and the value-scores then integrate into the ranking function $F$ that assigns a score to every tuple $t$ in $N$ (given by Equation 1):

$$\text{score}(t) = \sum_{i=1}^{m} w_i \cdot a_i$$

where $w_i$ represents the weight of attribute $A_i$ and $a_i$ represents the value score associated with $A_i$’s value in tuple $t$.

This inferred ranking function $F$ along with the user ($U$) and the query ($Q$) are loaded into the workload that forms the nucleus over which the second component of our framework is built. At the time of query ($Q'$), the user’s ($U'$) browsing choices may not always be available and hence the ranking function ($F_{U':Q'}$) cannot be derived. To address this, the second component (Inferring via Similarity Model) is formulated for deriving an appropriate ranking function (to rank $Q'$’s results for $U'$). This model takes as input, the user-query combination and determines, from the workload, a ranking
function derived for the most similar query (to $Q'$) given by the most similar user (to $U'$). This is achieved using the two individual metrics of – user similarity and query similarity. Based on the function derived, $Q'$’s results are ranked using Equation 1 and displayed to the user ($U'$).

4 Browsing Choices For Inferring Ranking Functions

In this section, we discuss our approach for inferring a ranking function for a query based on an user’s browsing choices. Since the ranking function is modeled as a linear weighted-sum function, we begin by explaining the learning technique that estimates the attribute-weights followed by the explanation of the normalization scheme used to derive the corresponding value-scores.

4.1 Learning Model For Deducing Attribute Weights

For a query $Q$ given by user $U$, we have at our disposal the set of query results $N$ generated by the database and the set $R$ ($\subset N$) of results selected by $U$ that matches his/her preferences. For an attribute $A_i$, the relationship between its values in $N$ and $R$ can aid in capturing the appropriate significance (or weight) associated by $U$ for $A_i$. In this paper, we propose a novel learning technique called the Probabilistic Data Distribution Difference method for estimating these attribute weights within a probabilistic framework.

**Probabilistic Data Distribution Difference:** Let us briefly revisit Example-1 where we hypothesize that $U_1$ is interested in ‘new’ and ‘red’ colored ‘Honda’ vehicles. For the sake of simplicity, let us focus only on the preferences for the “Color” attribute. The set $R$ will contain only ‘red’ colored vehicles whereas $N$ will contain vehicles with a wide range of colors. If we plot the probability distributions for the values of attribute “Color” in the sets $N$ (Figure 2-a) and $R$ ((Figure 2-b) respectively, it is evident that there is a significant difference between these distributions. This large difference indicates that for the results selected by $U_1$ in $R$, the “Color” values span a much smaller range as compared to the values for “Color” in $N$, thus signifying that this attribute is of definite importance.

In contrast, let us assume that the attribute “Mileage” is not of interest to $U_1$ and hence, the difference in probability distributions between the values for “Mileage” in the sets $R$ (Figure 3-b) and $N$ (Figure 3-a) would be relatively small.

Based on this analysis, we can conclude that – for an attribute $A_i$, if the difference in the probability distributions between $N$ and $R$ is large, it indicates that $A_i$ is important to the user, and hence, will be assigned a higher weight. In contrast, if the difference between the corresponding probability distributions is small, then $A_i$ will be assigned a lower weight. The next challenge is to translate these basic intuitions into a principled and quantitatively describable metric that signifies an appropriate attribute-weight. A widely popular measure, in the area of Image Processing is the Bhattacharya distance.
If the probability distributions for the values (in the same domain $X$) in set $N$ and $R$ is $N_p$ and $R_p$ respectively, then the distribution difference between $N$ and $R$, that in turn, determines the weight ($W_{A_i}$) of $A_i$, is given as:

$$W_{A_i} = D_B (N_p, R_p) = -ln (BC(N_p, R_p))$$  \hspace{1cm} (2)

where $BC$ (Bhattacharyya coefficient) for categorical and numerical attributes is respectively represented in Equations 3 and 4 as:

$$BC(N_p, R_p) = \sum_{x \in X} \sqrt{N_x, R_x}$$  \hspace{1cm} (3)

$$BC(N_p, R_p) = \int \sqrt{N_x, R_x} \, dx$$  \hspace{1cm} (4)

Using the above equations, we estimate the attribute-weight for every attribute in $A$ by calculating the corresponding Bhattacharyya distance for each attribute based on the distribution of its values in $N$ and $R$.

### 4.2 Normalization For Determining Value Scores

In order to rank the results of a query (using Equation 1), it is necessary that the score associated with every tuple is on a homogeneous scale. Although the attribute-weights estimated using the Bhattacharyya Distance (Section 4.1) are consistent (between $[0.0,1.0]$); due to the heterogenous nature of the attributes, the corresponding
values in the tuples have different types and ranges. Hence, it becomes mandatory to normalize these values.

**Normalizing Categorical Attributes:** We normalize categorical attributes (e.g., “Make”, “Color”, etc.) using a frequency-based approach. Given a query $Q$ by the user $U$, let $N$ and $R$ be the respective sets of query results and relevant results chosen by the user. Let $x$ be the value of a categorical attribute $A_i$. The the normalized value-score ($x_n$) is given as the frequency of the occurrence of $x$ in the set $R$, i.e., if there exists $r$ tuples in $R$ and $x_f$ is the frequency of $x$ in $R$, then the normalized value for $x$ is given by Equation 5.

$$x_n = \frac{x_f}{r} \quad (5)$$

**Normalizing Numerical Values:** In the case of numerical attributes such as “Mileage” (or “Price”), although an order can be established, it is not possible to determine whether a high value of “Mileage” should be ranked higher than a lower value or vice versa. To solve this problem, consider a scenario where the mean of the values for Mileage selected by $U$ in $R$ is lesser than the overall mean for the Mileage values in $N$. This indicates that $U$ selected vehicles with comparatively lesser mileage, and hence based on his/her preference, lesser-mileage values should be associated a higher value-score. Formally, for a numerical attribute $A_j$ –

- **Case-1:** If the mean for all values in $R$ is less than or equal to the mean for all values in $N$, a smaller value in $N$ should be assigned a higher normalized value.

- **Case-2:** If the mean for all values in $R$ is greater than the mean for all values in $N$, a larger value in $N$ should be assigned a higher normalized value.

Once, the order of numerical values is established, the next step is to actually determine the value-score. Toward that, we use the normalization-by-rank scheme. For a value $y$ in the domain of $A_j$ over the set $N$ (comprising of $n$ tuples), let –

- $V_l =$ number of tuples in $N$ whose value $<=$ $y$.
- $V_m =$ number of tuples in $N$ whose value $>$ $y$.

Then, the normalized value for $y$ is given by Equation 6.

$$y_n = \begin{cases} \frac{V_l}{n} & \text{Case-1} \\ \frac{V_m}{n} & \text{Case-2} \end{cases} \quad (6)$$

After this process of estimating value-scores, we have a set of normalized query results $N_n$ that can be used, along with the attribute weights, for estimating the score for every tuple. After such a ranking function for an user query is derived, it is loaded into a workload that acts as the core over which our proposed Similarity Model, which we explain in the next section, is established.
5 Similarity Model for Inferring Ranking Functions

For a Web database, it is impossible to procure the browsing choices for every user across the results of every query. In order to derive the ranking functions for such user queries, we are proposing the use of a Similarity Model that is based on the proposed notions of query- and user-similarity.

5.1 Query Similarity

For a single user ($U_1$), let $N_1$ be the results for a query ($Q_1$) for which no ranking function exists. Consider $U_1$’s workload that contains a set of queries – {$Q_2, Q_3, ..., Q_r$}, and let {$F_{12}, F_{13}, ..., F_{1r}$} be the ranking functions derived for each of these individual queries. Based on this information, it is useful to infer if any of the previously derived ranking functions can be used for ranking the results of $Q_1$. As alluded to in Example-2, the same user may have different ranking functions for different queries. Consequently, we cannot randomly choose a ranking function from $U_1$’s workload and use it to rank $N_1$. Hence, in order to determine an appropriate function, we introduce the notion of query similarity. We advance the theory that if $Q_1$ is most similar to a past query $Q_j$, then the ranking function ($F_{1j}$) derived for $Q_j$ can be used to rank the results of $Q_1$. In order to translate this proposal into a principled approach, we introduce two independent metrics for establishing similarity between queries – i) query-condition similarity, and ii) query-result similarity.

5.1.1 Query-Condition Similarity

We estimate the similarity between two queries by analyzing the relationship between the attribute values in the respective query conditions. We motivate this process with an intuitive scenario and follow it up with the formal description of the process.

Consider Example-1 with user $U_1$ asking query $Q_1$ (“Make=Honda And Location=Dallas,TX”). Let us reckon that $U_1$’s workload comprises of two queries – $Q_2$ (“Make=Toyota And Location=Atlanta,GA”), and $Q_3$ (“Make=Lexus And Location=Basin,MT”) for which the ranking functions $F_{12}$ and $F_{13}$ have been derived apriori. Intuitively, “Honda” and “Toyota” are vehicles with similar characteristics i.e., they have similar prices, mileage ranges, comfort levels, colors, and so on. In contrast, “Honda” is a very different car from “Lexus”. Hence, we can speculate that $U_1$’s preferences while selecting “Honda” would be similar to the ones displayed while looking for “Toyota”, and dissimilar to the ones shown while searching for “Lexus”. Similarly, when searching for vehicles in larger cities such as “Dallas” and “Atlanta”, $U_1$ may display similar preferences (such as vehicles with ‘alarm systems’, ‘secure door locks’, etc.) than the ones shown while searching in small towns such as “Basin”.

From the above analysis, we can infer that $Q_1$ is more similar to $Q_2$ than $Q_3$. It is also evident that in order to establish this similarity, we analyzed the relationship between the different values for each attribute in the respective query conditions. This intuitive analysis forms the basis for the notion of query-condition similarity, formally defined as:
Definition Consider two queries – \( Q \) and \( Q' \), each with the conjunctive selection conditions of the form “WHERE \( X_1 = x_1 \) AND \( \cdots \) AND \( X_s = x_s \)” and “WHERE \( X_1 = x'_1 \) AND \( \cdots \) AND \( X_s = x'_s \)” respectively. The query-condition similarity between \( Q \) and \( Q' \) is represented as the product of the similarities between the values \( x_i \) and \( x'_i \) for every attribute \( X_i \), and is shown in Equation 7:

\[
\text{similarity}(Q, Q') = \prod_{i=1}^{s} \text{sim}(X_i = x_i, X_i = x'_i)
\] (7)

In order to determine the RHS of the above equation, it is necessary to translate the intuition for similarity between values (e.g., “Honda” is more similar to “Toyota” than “Lexus”) to a formal model. We establish this value-level similarity between the different values of an attribute by determining the similarity between the results obtained from distinct queries which contain only these attribute values in their selection conditions. For instance, consider the values “Toyota” and “Honda” for the attribute “Make”. We generate two distinct queries (\( Q_T \) and \( Q_H \)) with the conditions – “Make = Toyota” and “Make = Honda” respectively, and obtain two individual sets of results – \( N_T \) and \( N_H \). By establishing a similarity between these two sets of results, we estimate the similarity between the two values. As mentioned earlier, since “Toyota” and “Honda” are similar vehicles, a large number of tuples in the results of these two sets will have similar values for attributes such as “Price”, “Mileage”, “Color”, etc. In contrast, since “Honda” and “Lexus” are dissimilar, there will be lesser number of tuples with similar values.

Formally, we establish the similarity between the values \( v_1 \) and \( v_2 \) for an attribute \( A_i \) as follows: We generate two queries – \( Q_{v_1} \) and \( Q_{v_2} \) with the respective selection conditions – “WHERE \( A_i = v_1 \)” and “WHERE \( A_i = v_2 \)”. Let \( N_{v_1} \) and \( N_{v_2} \) be the respective set of results obtained from the database for these two queries. The similarity between \( v_1 \) and \( v_2 \) is then given as the similarity between \( N_{v_1} \) and \( N_{v_2} \). To determine this result-level similarity, we use the variant of the cosine-similarity model proposed in [6] for addressing the empty-answers problem in relational databases.

Consider a tuple \( T = < t_1, t_2, \ldots, t_m > \) in \( N_{v_1} \) and a tuple \( T' = < t'_1, t'_2, \ldots, t'_m > \) in \( N_{v_2} \). The similarity between \( T \) and \( T' \) is given by Equation 8.

\[
\text{sim}(T, T') = \sum_{i=1}^{m} \text{sim}(t_i, t'_i)
\] (8)

where

\[
\text{sim}(t_i, t'_i) = \begin{cases} 
1 & \text{if } t_i = t'_i, \\
0 & \text{if } t_i \neq t'_i.
\end{cases}
\] (9)

Equation 9 will work improperly for numerical attributes where exact matches are difficult to find across tuple comparisons. In this paper, we assume that numerical data has been discretized into categories using a meaningful scheme. The problems occurring in such categorization are beyond the scope of this paper and as explained in [5], there exists a body of work that addresses this challenge.

Based on Equation 8, we determine a similarity between \( T \) and every tuple in \( N_{v_2} \) (i.e., \( T'_1, \ldots, T''_m \)) and the maximum individual value of similarity obtained is assigned as
the similarity between the tuple $T$ and $N_{v_2}$ given by $- \text{sim}(T, N_{v_2})$. The similarity between the two sets $N_{v_1}$ and $N_{v_2}$ is then determined as the average of all similarity values between the tuples in $N_{v_1}$ with the set $N_{v_2}$ and is given by Equation 10.

$$\text{sim}(N_{v_1}, N_{v_2}) = \frac{\sum_{i=1}^{n} \text{sim}(T_i, N_{v_2})}{n}$$ (10)

Thus, by substituting the values for value-similarity in Equation 10 into the RHS of Equation 7, we can estimate the similarity between a given pair of queries.

5.1.2 Query-Result Similarity

In this metric, the similarity between two queries is determined by comparing their results. In the previous Section, we established similarity between values by computing the similarity of the results generated in response to queries containing these values. The objective there was based on the intuition that users’ ranking preferences depend solely on the queries. We provide the motivation behind this metric with an intuitive scenario and follow it up with the formal description of the process.

Consider the user $U_1$ from Example-1 asking the query $Q_1$ ("Make=Honda and Location=Dallas,TX"). We have speculated that $U_1$ is interested only in ‘new’ and ‘red’ colored vehicles. Let us assume that $N_1$ contain only ‘red’ colored vehicles. In that case, the ranking preferences and hence the ranking function for $U_1$ is likely to change. Assume a query $Q_4$ ("Make=Honda and Color=Red and Location=Houston,TX") in $U_1$’s workload (for which a ranking function $F_{14}$ has been derived) whose query results $N_4$ are similar to the ones obtained for $Q_1$ (i.e., set $N_1$). Now, it is likely that $U_1$ would display similar preferences over $N_1$ as shown over $N_4$. In other words, the ranking preferences of the user are solely decided by the results displayed to him/her. Based on this, we propose the metric of query-result similarity which is defined as:

**Definition** Consider two queries – $Q$ and $Q'$, each with the conjunctive selection conditions of the form “WHERE $X_1 = x_1 \text{ AND} \cdots \text{ AND} X_s = x_s$” and “WHERE $X_1 = x'_1 \text{ AND} \cdots \text{ AND} X_s = x'_s$” respectively. Let $N$ and $N'$ be the set of results obtained for these queries from the database. The query-result similarity between $Q$ and $Q'$ is estimated as the similarity between the result sets $N$ and $N'$, given by Equation 11:

$$\text{similarity}(Q, Q') = \text{sim}(N, N')$$ (11)

The similarity between the $N$ and $N'$ is determined using the Equations 8, 9 and 10 explained in Section 5.1.1

5.1.3 Which Query Similarity?

In the previous two Sections, we have proposed two independent notions of query similarity. The logical question that arises is – which one is better? Intuitively, it is evident that in the case of most users, ranking preferences are predetermined at the time of query and hence, when queries tend to be similar, their corresponding ranking functions would also be similar. This leads us to believe that query-condition similarity seems to be a more intuitive solution for addressing the notion of query similarity.
The metric of *query-result* similarity comes into play when the ranking preferences in the user’s mind cannot be applied on the query results (when all or none of the results satisfy the user’s preferences). In these situations, the user’s preferences are driven by the results shown to him and for such set of queries, this metric tends to be more effective. In a more general setting, it appears that *query-condition* similarity is applicable in the context of *Many Answers problem* [5], whereas *query-result* similarity seems appropriate in the domain of *Empty Answers problem* [6]. We are currently investigating techniques to prove this hypothesis.

Although we do not address the issue of efficiency in this paper, at the outset it is evident that *query-condition* similarity is more desirable since similarities between value-pairs for different attributes can be pre-computed. The corresponding storage requirements would also be minimal. In contrast, in the case of *query-result* similarity, it will not be possible to pre-compute similarity between every pair of query results since queries cannot be predicted beforehand. Hence, it would be necessary to store query results and determine the similarity at runtime - a costly process.

This, our analysis indicates that the former metric, being more elegant and intuitive seems to be the better choice in the current context. This claim is further validated by our experimental results (Section 6).

### 5.2 User Similarity

Consider the user $U_1$ from *Example-1* asking the query $Q_1$ (“Make=Honda And Location=Dallas,TX”) for whom, in the absence of browsing choices, we need to determine a ranking function ($F_{11}$). Let us presume that there exist a set of users {$U_2$, $U_3$, ..., $U_r$} in the workload who have each asked $Q_1$, and for whom ranking functions {$F_{21}$, $F_{31}$, ..., $F_{r1}$} have been derived based on their browsing choices. In such a case, it is useful to investigate if any of these functions can be applied to rank the results of $Q_1$ for $U_1$. However, as suggested in *Example-1*, different users may have different ranking functions for the same query. Therefore, we cannot randomly choose a ranking function and use it to rank $N_1$ for $U_1$. To deal with this challenge, we introduce the notion of *user similarity* which proposes that – if $U_1$ is most similar to an user $U_i$, then the ranking function ($F_{i1}$) derived for $U_i$ can be used to rank the results of $Q_1$.

Consider a snapshot of the workload information for $U_1$ and $U_2$ shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>$Q_2$</th>
<th>$Q_3$</th>
<th>$Q_4$</th>
<th>$Q_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_1$</td>
<td>$F_{12}$</td>
<td>$F_{13}$</td>
<td>–</td>
<td>$F_{15}$</td>
</tr>
<tr>
<td>$U_2$</td>
<td>$F_{22}$</td>
<td>$F_{23}$</td>
<td>$F_{24}$</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 1: Sample Workload for Users – $U_1$ and $U_2$
functions exist for both users, and the similarity is then expressed as the combined similarity between the ranking functions derived for these queries. Formally,

**Definition** Consider two users – $U_1$ and $U_2$ asking the same set of queries – $\{Q_1, Q_2, ..., Q_r\}$ for which ranking functions ($\mathcal{F}_{1i}, \mathcal{F}_{12}, ..., \mathcal{F}_{1r}$) and ($\mathcal{F}_{2i}, \mathcal{F}_{22}, ..., \mathcal{F}_{2r}$) have been derived based on their browsing choices. Then, the user similarity between $U_1$ and $U_2$ is expressed as the average similarity between the individual ranking functions derived for $U_1$ and $U_2$ for each query $Q_j$, and is shown in Equation 12:

$$
similarity(U_1, U_2) = \frac{\sum_{j=1}^{r} sim(\mathcal{F}_{1j}, \mathcal{F}_{2j})}{r}
$$

In order to determine the RHS of the above equation, it is necessary to quantify a measure that establishes similarity between two ranking functions ($\mathcal{F}_{1j}$ and $\mathcal{F}_{2j}$). We estimate this function similarity as the value of the *Spearman’s rank correlation coefficient* ($\rho$) between the ranked result sets i.e., the sets obtained by applying these ranking functions on the individual query results. We choose the *Spearman coefficient* for comparing the two ranked sets based on the results of the survey [20] that proves its usefulness in comparing ranked lists with respect to other proposed metrics.

Consider two functions $\mathcal{F}_{1j}$ and $\mathcal{F}_{2j}$ derived for a query $Q_j$ with results $N_j$. We apply each of these functions individually on $N_j$ to obtain two ranked set of results – $N_{R1j}$ and $N_{R2j}$. If the total number of results in each of these ranked sets is $N$, and $D$ is the difference between the ranks of corresponding values in $N_{R1j}$ and $N_{R2j}$, then we express the similarity between the two functions $\mathcal{F}_{1j}$ and $\mathcal{F}_{2j}$ as the *Spearman’s rank correlation coefficient* ($\rho$) given by Equation 13:

$$
sim(\mathcal{F}_{1j}, \mathcal{F}_{2j}) = 1 - \frac{6 \times \sum_{i=1}^{N} D^2}{N^2 \times (N^2 - 1)}
$$

For each pair of ranking functions, that exist in the workload of users $U_1$ and $U_2$ over the set of commonly asked queries, the *Spearman’s rank correlation coefficient* calculated from Equation 13 is fed into the RHS of Equation 12 to determine the user similarity between $U_1$ and $U_2$. Once a similarity is established between the user ($U_1$) and every other user – $\{U_2, U_3, ..., U_p\}$ for whom a ranking function exists for $Q_1$, the results of $Q_1$ for $U_1$ are ranked by applying the ranking function derived for that user $U_i$ who is most similar to $U_1$.

### 5.3 The Similarity Model

In order to derive an user’s ($U_1$) ranking function for a query ($Q_1$), we have proposed two independent approaches based on user- and query-similarity. In a general context of Web database applications, it is highly unlikely that there exists a query $Q_j$ in $U_1$’s workload that is very similar to $Q_1$, thus hampering the applicability of only query similarity. Similarly, the likelihood of finding an user $U_i$, very similar to $U_1$, and for whom a ranking function exists for $Q_1$ is rare, thus, reducing the potential of achieving good ranking via only user similarity. In a more generic setting, the best a workload can achieve would be to contain the ranking function for a similar query (to $Q_1$) derived
Input: Query $Q_1$, User $U_1$, Workload $W$
Output: Ranking Function $F_{ij}$
1. foreach $U_i (\in U_{set} = \{U_1, \ldots U_p\})$ in $W$ do
   | Calculate user-similarity($U_1, U_i$)
end
2. foreach $Q_j (\in Q_{set} = \{Q_1, \ldots Q_r\})$ in $W$ do
   | Calculate query-similarity($Q_1, Q_j$)
end
3. sort($U_{set}$) // descending order
4. sort($Q_{set}$) // descending order
5. Initialize matrix $F$ – $U_{set}$ as rows & $Q_{set}$ as columns
6. foreach $U_i$ & $Q_j$ do
   | 7. if $F_{ij} \in W$ then
       |     $F[i][j] = F_{ij}$
   | else
       |     $F[i][j] = \text{null}$
end
8. $F_{ij} = \text{Get-RankFn}(F)$
9. return $F_{ij}$

Algorithm 1: Inferring Ranking Functions using Similarity

for a similar user (to $U_1$). In order to determine such a function, we combine the two measures into a single Similarity Model. The goal of this model is to determine a ranking function ($F_{ij}$) derived for the most similar query ($Q_j$) to $Q_1$ given by the most similar user ($U_i$) to $U_1$ to rank $Q_1$’s results. The process for finding such an appropriate ranking function is represented by Algorithm 1.

The input to the algorithm is the user ($U_1$) and the query ($Q_1$) along with the workload ($W$) containing ranking functions for past user queries. The algorithm begins by determining a user similarity between $U_1$ and every user $U_i$ (Step 1), and a query similarity between $Q_1$ and every query $Q_j$ (Step 2) from the workload. Based on these similarity calculations, we assign a rank to every query and user based on their similarity with $Q_1$ and $U_1$ respectively, such that a query very similar to $Q_1$ gets a higher rank than the one less similar to $Q_1$ (Steps 3 and 4). Based on these ranked sets, we establish a $p \times r$ matrix $F$ where the rows represent individual users from the ranked set $U_{set}$ and the columns indicate the queries from the ranked set $Q_{set}$ (Step 5). For instance, the matrix shown in Table 2 represents the queries ($Q_1$ ... $Q_4$) as columns such that $Q_2$ is more similar to $Q_1$ than $Q_3$ which, in turn, is more similar to $Q_1$ than $Q_4$. The users ($U_1$ ... $U_5$) are similarly represented as rows such that $U_2$ is more similar to $U_1$ than $U_3$ and so on.

Every cell ($F[i][j]$) in this matrix represents a ranking function derived for a query $Q_j$ given by an user $U_i$. Steps 6 and 7 of the algorithm searches the workload for the existence of a function for every such cell and populates the matrix. As shown in the matrix in Table 2, cells for which ranking functions exist (e.g., $F[2][2]$) are
Table 2: Matrix for Inferring Ranking Function

<table>
<thead>
<tr>
<th></th>
<th>(Q_1)</th>
<th>(Q_2)</th>
<th>(Q_3)</th>
<th>(Q_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(U_1)</td>
<td>?</td>
<td>null</td>
<td>null</td>
<td>(F_{14})</td>
</tr>
<tr>
<td>(U_2)</td>
<td>null</td>
<td>(F_{22})</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>(U_3)</td>
<td>null</td>
<td>null</td>
<td>(F_{32})</td>
<td>null</td>
</tr>
<tr>
<td>(U_4)</td>
<td>(F_{41})</td>
<td>null</td>
<td>null</td>
<td>(F_{44})</td>
</tr>
<tr>
<td>(U_5)</td>
<td>(F_{51})</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

populated whereas empty cells reflect that no ranking functions have been derived for the particular user-query combination (e.g., \(F_{21}\)). From this populated matrix, the similarity model needs to derive a ranking function that can be used to rank the results of \(Q_1\) for \(U_1\). The ‘Get-RankFn’ function (Step 8) aids this process by determining a cell (\(F[i][j]\)) in the matrix that satisfies both of the following two conditions:

1. **Condition-1**: \(F[i][j] \neq \text{null}\), and
2. **Condition-2**: rank(\(U_i\)) + rank(\(Q_j\)) is minimum

The former condition is self-explanatory since we need to find a cell for which a function has been derived apriori. The second condition ensures that the Similarity Model determines the function that belongs to the most similar query (to \(Q_1\)) derived for the most similar user (to \(U_1\)).

Table 3: Matrix Populated With Sum of Ranks

<table>
<thead>
<tr>
<th></th>
<th>(Q_1)</th>
<th>(Q_2)</th>
<th>(Q_3)</th>
<th>(Q_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(U_1)</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>(U_2)</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>(U_3)</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>(U_4)</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>(U_5)</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

For the matrix shown in Table 2, to determine if a user-query combination satisfies **Condition-2**, consider the matrix shown in Table 3 where every cell \(F[i][j]\) represents the sum of the ranks derived from \(U_i\) and \(Q_j\). For the cell \(F[2][2]\), the value is determined by summing the rank of \(Q_2\) (= 2 in the ranked set \(Q_{set}\)) with the rank of \(U_2\) (= 2 from the ranked set \(U_{set}\)). Therefore, the value in this cell equates to 4. By observing the matrices in Tables II and III respectively, it is evident that the cell \(F[2][2]\) satisfies both the above conditions, and hence the function \(F_{22}\) is most appropriate for ranking the results of \(Q_1\).

In order to determine such a cell, the ‘Get-RankFn’ function needs to traverse through the matrix, and such traversal can be accomplished, since the rows and columns
are arranged in their ranked order of similarity with $U_1$ and $Q_1$, using a simple breadth-first-search (or BFS) algorithm or any contour-line algorithm. In the interest of space, we do not discuss the details of these algorithms. Based on the cell at which the ‘Get-RankFn’ function terminates, the ranking function $F_{ij}$ for this cell is selected (Step 9) by the Similarity Model to rank the results of $Q_1$ for $U_1$.

6 Experiments

In this section, we report the results of an experimental evaluation for the proposed ranking models. We mention at the outset that preparing an experimental setup was extremely challenging, as unlike Information Retrieval, there are no standard benchmarks available, and we had to conduct user studies to evaluate the rankings produced by our proposed models.

6.1 Experimental Setup

The Web database used for our experiments is obtained from the publicly available Google Base’s vehicle database that provides APIs for issuing queries and obtaining the corresponding results. The schema exported by Google Base comprises of 8 categorical attributes – Make, Model, Vehicle-Type, Mileage, Price, Color, Location, Transmission. Although the attributes ‘Price’ and ‘Mileage’ are numeric, they are categorized apriori by Google into meaningful ranges. In addition, for every attribute, the domain of values across which query conditions can be formed (e.g., ‘Chevrolet’, ‘Honda’, ‘Toyota’, ‘Volkswagen’, ... for the ‘Make attribute) is provided. For every query asked via the API, Google Base returns a maximum of 1000 results in an XML format that is translated into a relational table for our experiments. Additionally, for every attribute, the frequency of occurrence of individual values in the result set is provided.

Currently, Google Base (or any other Web database) does not provide the statistics regarding users, their queries and their respective browsing choices over the query results. Since these are important components in our ranking framework, we had to rely on user studies for obtaining some of these statistics. To that end, we generated a pool of 60 random queries, each comprising of conditions based on randomly selected attributes and their values from the Google Base schema. From this set, we manually selected 18 meaningful queries that represent the kind of queries generally formulated by Web users. Table 4 shows 5 of such queries, and the reader is directed to http://129.107.12.239/rankSurvey to check the full list of selected queries. These queries were then translated into Google API format (e.g., translating numerical values into categorical ranges defined by Google) for obtaining the required set of results.

In order to obtain the necessary user statistics, we conducted a survey where every user was shown 18 queries (one-at-a-time) along with their individual results obtained from Google Base. The idea was to capture the users’ browsing choices for the displayed query and its results. However, we believe that asking a user to select relevant tuples by browsing through the results for every query (across 18 such queries) would...
Table 4: Sample Queries Used In Experimental Evaluation

| Q1  | “Make=Honda, Location=Dallas,TX, Price<10,000$” |
| Q2  | “Make=Toyota, Location=Miami,FL, Price<8,000$” |
| Q3  | “Make=Chrysler, Location=Dickinson,ND, Price<35,000$” |
| ... | ... |
| Q10 | “Location=Chicago,IL, Mileage<100,500, Year>2004” |
| Q11 | “Location=Detroit,MI, Mileage <175,000, Year>2002” |
| ... | ... |

be a tedious task. Hence, in our survey, we narrowed the users’ task to simply expressing their ranking preferences in terms of attributes i.e., for every displayed query and its results, the user had to assign a preference (on a scale of 1 to 10 with 10 being highest) for every attribute that they would consider important for the given query. Thus, due to the difficulties in obtaining preferences for attribute values, in this evaluation, we focus on establishing the quality of rankings with respect to only attribute-weights. For the rest of the section, we use the terms attribute-weights and ranking functions interchangeably. The survey was taken by 60 users (mostly graduate students and faculty members) who provided attribute-weights for most of the queries displayed to them.

We now explain the individual evaluation results obtained for our ranking models.

6.2 Learning Model Evaluation

In this setting, we test the quality of our proposed learning method (Probabilistic Data Distribution Difference) in deriving attribute-weights for an user query. Since this method relies on the availability of the set of results chosen by user, we need a mechanism that can generate these relevant results from the preferences provided by the user.

Consider the query $Q_1$, shown in Table 4 for which an user (say $U_1$) provides the set of attribute-weights ($W_1$). Using this set of weights, we rank the query’s results ($N_1$) to obtain a ranked set of results ($N_{r1}$). From this ranked set, we select a set of 25 tuples that we represent as the set ($R_1$) of tuples chosen by $U_1$ for $Q_1$’s results. The reason behind choosing 25 tuples is based on the intuition that a Web user, in general selects a very small percentage of users shown to him/her. We tested our model with set $R$ varying from 10 to 50 tuples, and since the results are similar, in the interest of space, we explain our results based on set $R$ comprising of 25 tuples.

The next logical question that arises is – how do we choose the 25 tuples from the ranked set of results ($N_{r1}$)? For this, we use sampling techniques to select the tuples. It is natural that based on users’ preferences, higher ranked tuples (from $N_{r1}$) should have a higher probability of being sampled than the lower ranked ones. Hence, the sampling technique we choose selects the tuples using power-law distributions. Specifically, we choose the popular – Zipf, Zeta, and Pareto power law probability distributions to achieve this sampling. Additionally, we also use a simple sampling scheme that selects
only the top-25 ranked tuples from $N_{r1}$.

Based on this derived set ($R_1$) and the original query results ($N_1$), our learning method derives the set of attribute-weights ($W_1'$). In order to establish the quality of this ranking function, we use the derived set of weights ($W_1'$) to rank the set $N_1$ and obtain a new set of ranked results $N_{r2}$. The quality of the ranking function is then estimated as the Spearman rank correlation coefficient (Equation 13) computed between the two ranked result sets – $N_{r1}$ and $N_{r2}$. Higher the value of this coefficient, better is the quality of the ranking function, and vice-versa. We compare the quality of the ranking function learnt using our learning method to the quality achieved by using two established and popular learning models – linear regression and naive bayesian classifier.

Figure 4: Ranking Quality using Learning Methods (top-25 Sampling)

Figure 5: Average Ranking Quality using Learning Methods (top-25 Sampling)

Figure 4 compares the ranking quality, for a randomly chosen user ($U_1$), achieved by the three learning models in deriving attribute weights for each individual query. In
In the previous setting, the top-25 result sampling was chosen to form the set $R_1$. In order to prove the generic effectiveness of our model, we establish the quality of ranking achieved by our method when the set $R_1$ is chosen using different sampling techniques (Figure 6). In this graph, each vertical bar represents the average ranking quality achieved across all queries and users by each learning model for each sampling technique. The graph in the figure clearly indicates that across different sampling techniques, our proposed method performs consistently better.

6.3 Similarity Model Evaluation

In this setting, we test the quality of ranking achieved when query results are ranked using the Similarity Model. The workload used for this setting comprised of the user-query pairs for which an user provided attribute-weights as part of the survey. In all, the workload comprises of 18 queries and 60 users, across which 974 ranking functions (or sets of attribute weights) were collected. We begin by displaying the ranking quality achieved by using individual measures of similarity, and then illustrate the quality achieved by the combined similarity model.

**Query Similarity:** Similar to the setting in Section 6.2, consider the weights ($W_1$) provided by $U_1$ for $Q_1$, from which we obtain a ranked set of results $N_{r1}$ from $N_1$. The similarity model determine a query $Q_j$ that is most similar to $Q_1$ using the metric of query-condition similarity (Section 5.1.1), and for which $U_1$ has provided a set of weights ($W_1$). We rank $N_1$ using $W_1$ to obtained a ranked set of results ($N_{r2}$). Similarly, we derive a query $Q_j$ using the notion of query-result similarity (Section 5.1.2). We apply $W_j$ (provided by $U_1$ for $Q_j$) to $N_{r1}$ and obtain another ranked set of results $N_{r3}$. The Spearman rank correlation coefficient (Equation 13) computed individually
between – i) \( N_{r1} \) and \( N_{r2} \), and ii) \( N_{r1} \) and \( N_{r3} \) establishes the quality of the ranking achieved using the two measures of query similarity.

The evaluation results for this metric are shown in Figure 7 which displays the ranking quality achieved for each query given by a randomly selected user (\( U_1 \)). In Figure 8, we show the average quality achieved across all users for every individual query. The graph demonstrates that query-condition similarity clearly outperforms the query-result similarity (which verifies the intuition proposed in Section 5.1.3). The bottom line plot in both the graphs represents the average ranking quality obtained using the remaining queries in the workload which is estimated as follows: For \( Q_1 \) we select the remaining set of weights (15, if user has provided functions for all 18 queries) and use them to individually rank \( N_1 \) to obtain 15 different ranked set of results. We then estimate the ranking quality as the average of the Spearman coefficient values obtained between \( N_1 \) and each of these 15 ranked sets.

**User Similarity:** Analogous to the above results, we evaluate the effectiveness of
our second proposed measure—user similarity. First, we consider the function given by $U_1$ for $Q_1$ from which we obtain a ranked set of results $N_{r1}$. Then, for the user $U_1$, we determine the most similar user $U_k$ who has provided a set of weights ($W_k$) for $Q_1$. We determine the quality of ranking as the Spearman coefficient computed between the two ranked result sets—$N_{r1}$ and $N_{r2}$, where $N_{r2}$ is obtained by ranking $N_1$ using $W_k$.

Figure 9 shows the ranking quality individually achieved for all queries for the user ($U_1$) using the user-similarity measure. In Figure 10, we show the average quality achieved across all users for every individual query. The bottom line plot in the graphs represents the average quality achieved using the remaining users which is determined as follows: We apply the set of weights (if available) provided by every other user for $Q_1$ to obtain a set of individual ranked sets, and determine the average quality as the one achieved by computing the Spearman coefficient between $N_1$ and each of these ranked sets.

**Combined Similarity:** We now present the quality of the ranking achieved by our
combined similarity model whose process for determining an appropriate function is illustrated in Algorithm 1. The random user ($U_1$) chosen so far has provided a function for every query; hence, the quality of ranking achieved using combined similarity is the same as the one obtained via query similarity.

In order to prove the effectiveness of this model, we chose two additional queries (say $Q_{19}$ and $Q_{20}$), such that they are both completely dissimilar (using both measures of query similarity) to the remaining 18 queries, and asked the users to provide attribute weights for these two queries. We carried out the process of determining the ranking quality for these two queries using only query similarity, and as expected, the ranking quality was extremely poor as seen in Figure 11.

Similarly, we created a dummy user (say $U_{61}$) who gave a single ranking function (i.e., same attribute-weights) for every query displayed to him (an anomaly in a user-dependent ranking framework). We carried out the process of determining the ranking quality for this users using only user similarity, and as expected, the ranking quality
was extremely poor and is shown in Figure 12.

However, in the case of a Combined Similarity Model, the contradictory results obtained in the above two cases would be resolved. For instance, for an user $U_i$ asking $Q_{19}$ in the first case, although there exists no similar query, the model would obtain a similar user who might have provided a function for $Q_{19}$. Analogously, since $U_{61}$ provided the same function for all queries, in a combined setting, although no user is similar to him/her, the model would determine the function of a similar query provided by $U_{61}$. This claim is verified in Figure 13 that displays the average ranking quality achieved by the combined similarity model across all 61 users for every individual query (for all 20 queries). As can be seen in the figure, this model can deal with anomalous users and queries elegantly and displays an overall consistent ranking quality.

6.4 Evaluation of the Complete Framework

In the previous two sections, we discussed the results obtained by evaluating each model in isolation; however, to establish the validity of our framework, it is necessary to evaluate the entire system as a whole. For such evaluation, actual statistics with regards to users, their queries, as well as their browsing choices would need to be collected. As alluded to earlier, currently no Web databases provide such statistics. A synthetic data generation mechanism could be used; but modeling different users synthetically and establishing browsing choices for different user queries is a problem beyond the scope of this paper. Our focus, in this experimental evaluation, however, was to establish that there is merit in using similarity models and browsing choices for inferring ranking functions which is visible from the results obtained so far.

\[\text{In the interest of clarity, in the graph, we did not plot the average quality across all non-similar ranking functions for a query.}\]
7 Conclusion

We proposed an automated solution to the problem of ranking query results for Web databases in a user- and query-dependent setting. Our models derive ranking functions by analyzing users’ browsing choices over query results, and by establishing similarity measures between users and queries. To that end, we proposed a novel learning method in addition to two new measures of similarity for deriving such ranking functions across different user queries. We presented results of preliminary experiments which demonstrate the quality of ranking achieved by our models.

Our work brings forth several intriguing challenges. For example, in this paper we did not discuss the aspect of efficiency associated with this framework since our primary objective was to establish a model for query- and user-dependent ranking. We are currently investigating different data structures and search algorithms for addressing this concern. Additionally, the seamless integration of our proposed ranking functions in well-established top-k retrieval algorithms is another aspect that we plan to investigate. Similarly, we are currently analyzing the generalization of our framework for range queries with different Boolean conditions over multiple data tables. Finally, comprehensive quality benchmarks in addition to real user and query statistics for database ranking need to be established. This would provide future researchers with a more unified and systematic basis for evaluating their retrieval algorithms.

References


